Algebra for Analytics:

Two pieces for scaling computations, ranking and learning

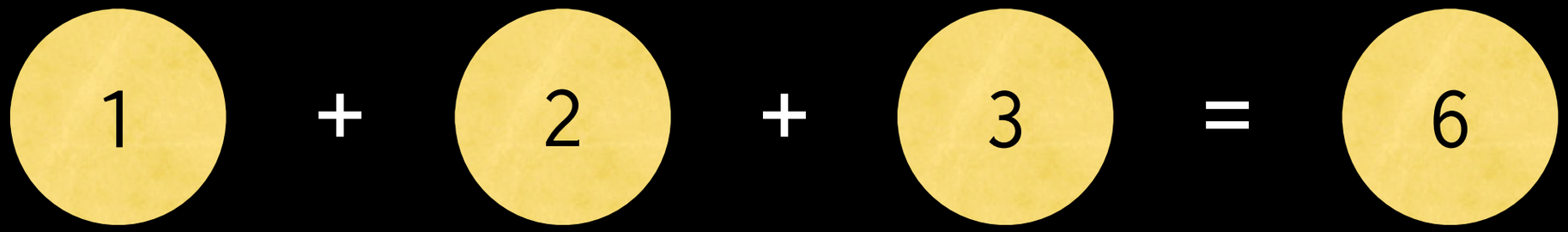
Strata, Santa Clara

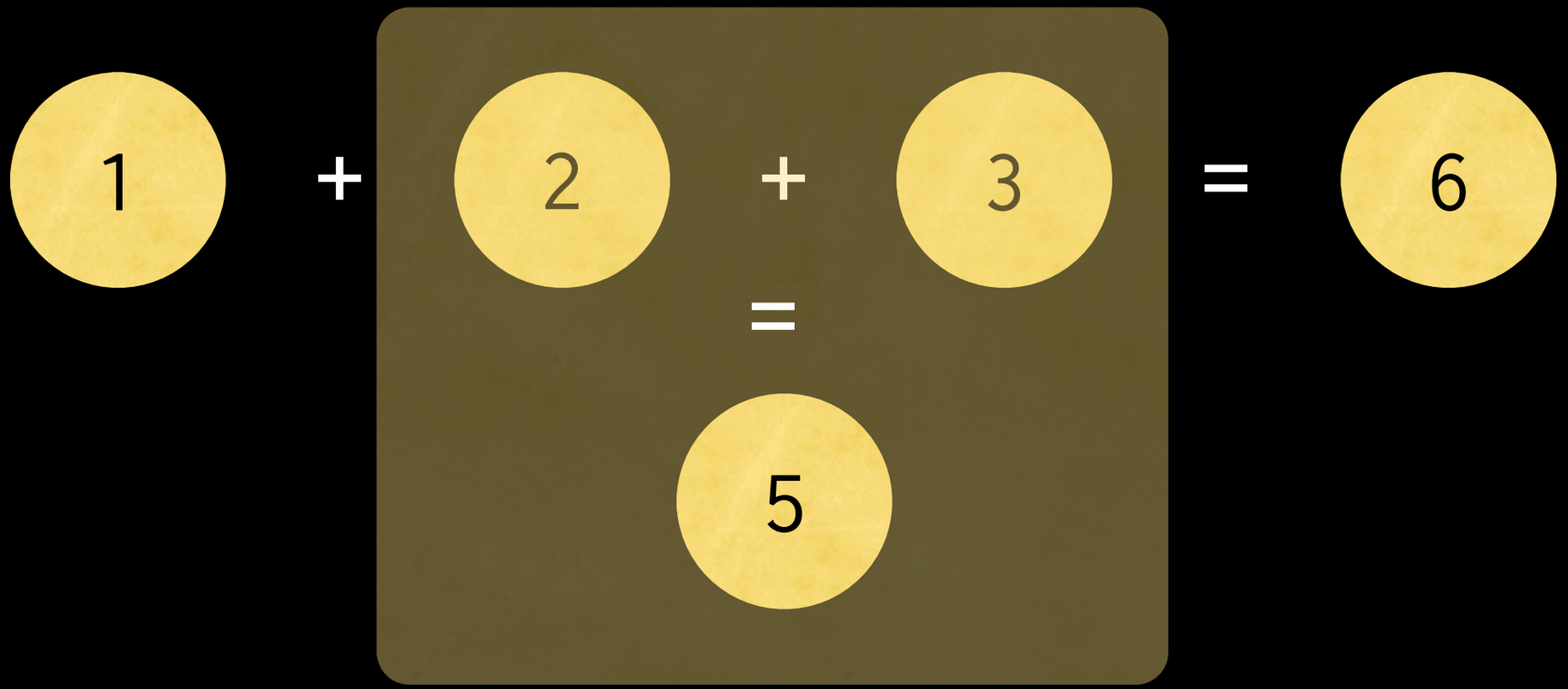
Who is this dude?

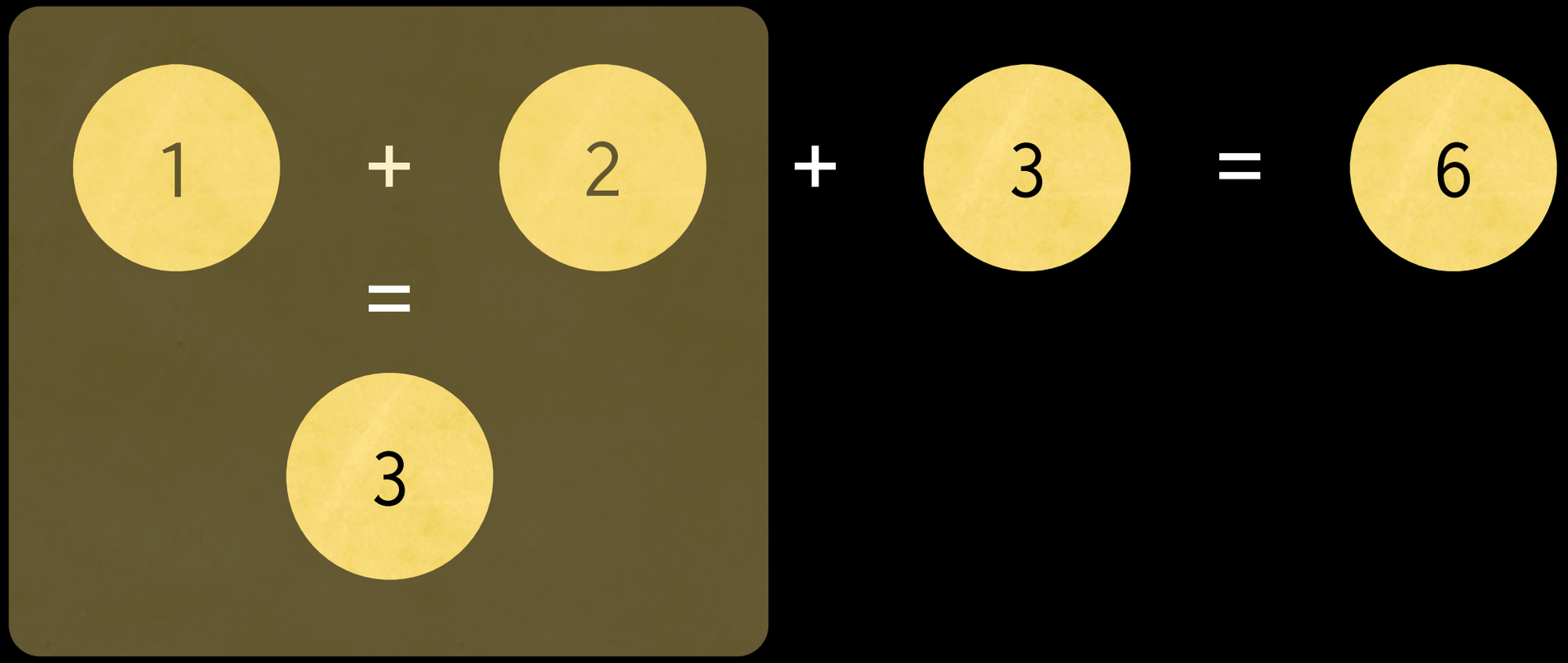
- Oscar Boykin @posco
- Staff Data Scientist at Twitter -- co-author of scala+hadoop library @Scalding -- co-author of realtime analytics system @Summingbird
- Former Assistant Professor of Electrical + Computer Engineering at Univ. Florida -- Physics Ph.D.

- Algebra (Monoids + Semigroups)
- Hash, don’t sample! (Bloom/HyperLogLog/Count-min)

Part 1: Algebra

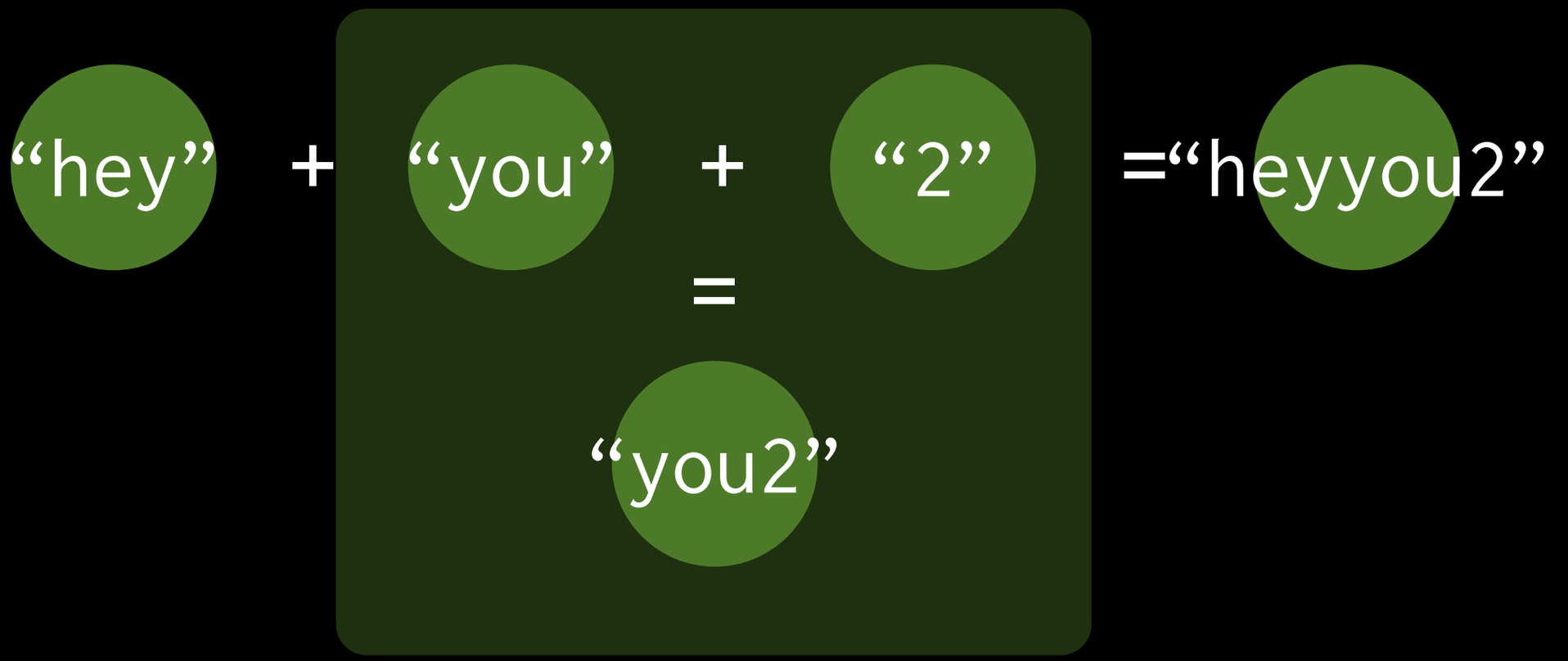

1
+
2
+
3
=
6

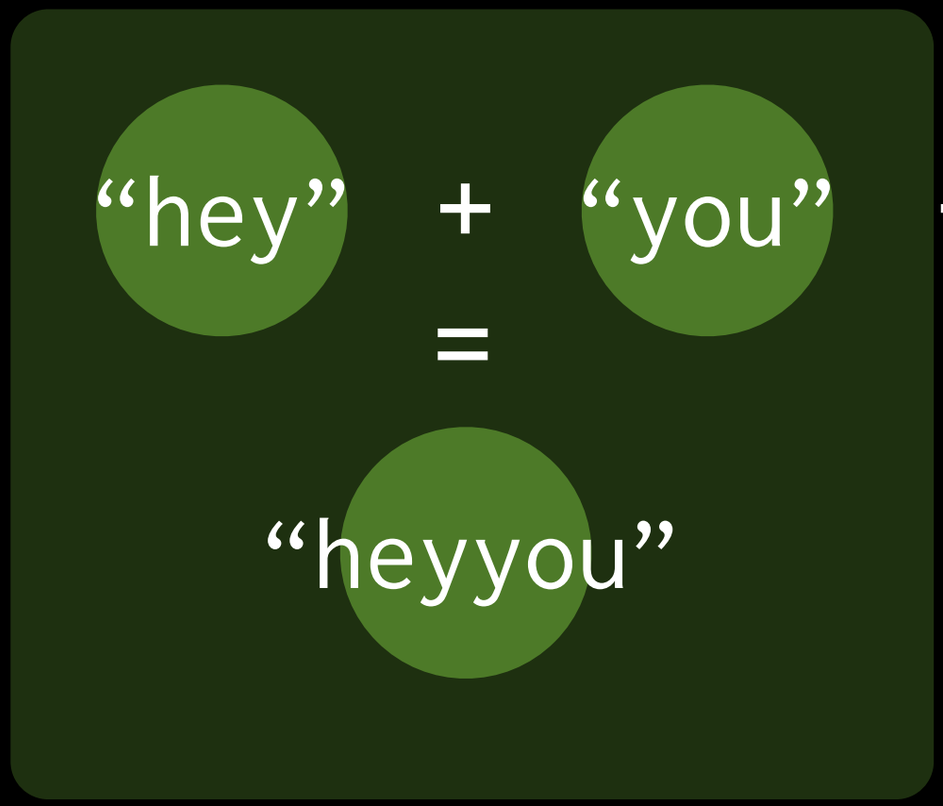

1
+
2
+
3
=
6
=
5


1
+
2
=
3
+
3
=
6

Associativity:

(a+b)+c = a+(b+c)


“hey”
+
“you”
+
“2”
=“heyyou2”
=
“you2”


“hey” + “you” + “2” =“heyyou2”
=
“heyyou”

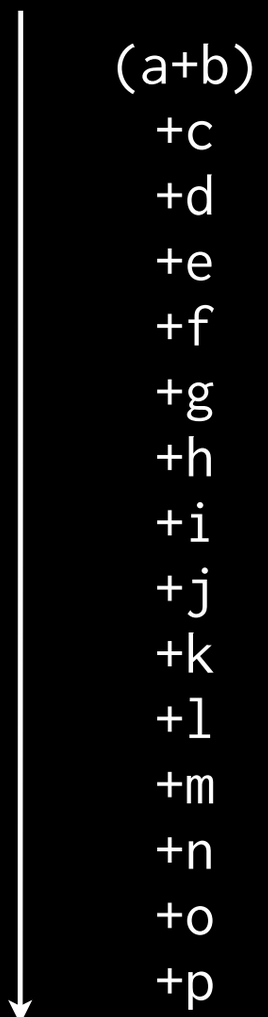
Associativity:

(a+b)+c = a+(b+c)

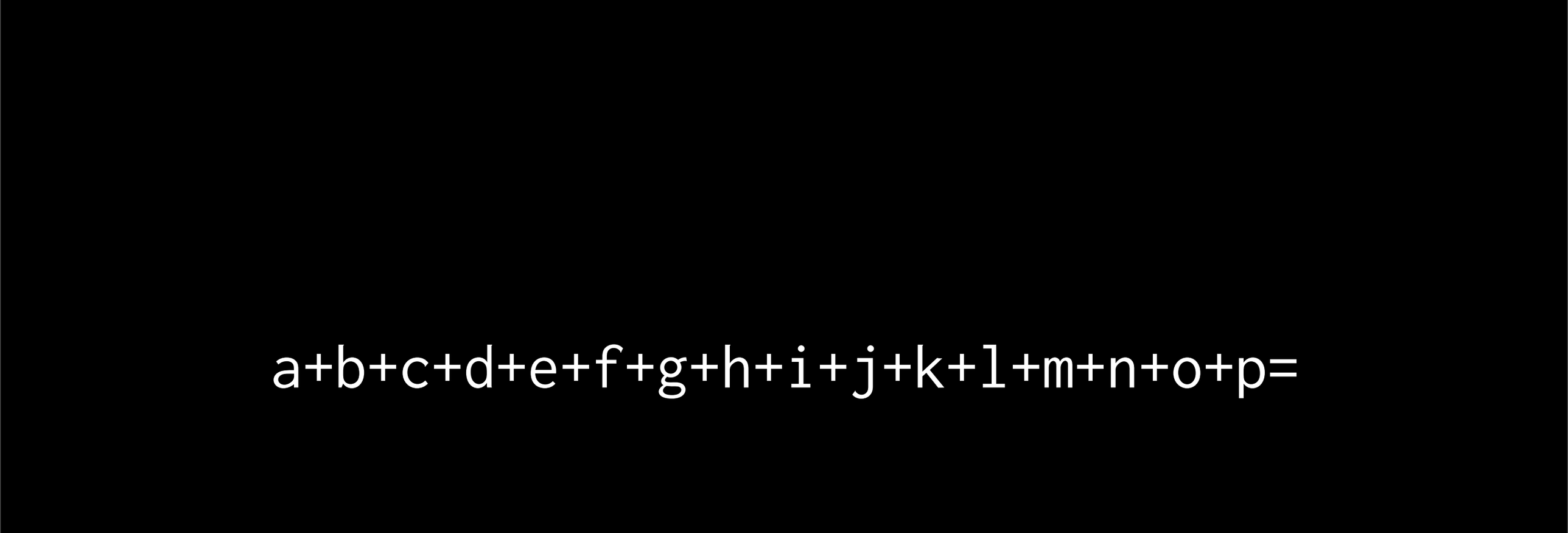
Let's you put ()
where you want!

a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p=

Latency = 15 =(n-1)

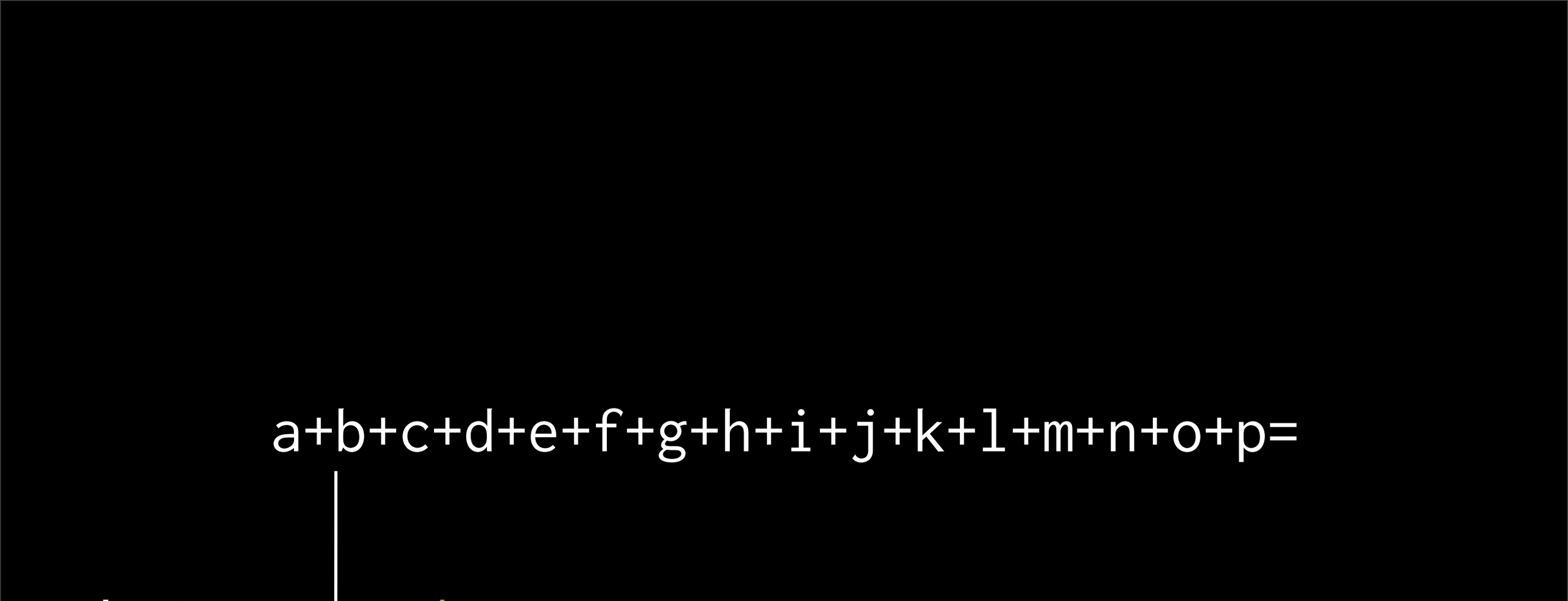


a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p=



a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p=

Latency = 4 =log_2(n)



Associativity allows parallelism in reducing!

Even without commutativity

But not everything has this structure!


YEAH, WELL THAT'S JUST,
LIKE,
YOUR OPINION, MAN

Example Monoids

- `(a min b) min c = a min (b min c)`
- `(a max b) max c = a max (b max c)`
- `(a or b) or c = a or (b or c)`
- `int addition: (a + b) + c = a + (b + c)`
- `set union: (a u b) u c = a u (b u c)`
- `harmonic sum: 1/(1/a + 1/b)`
- `and vectors: [a1, a2] max [b1, b2] = [a1 max b1, a2 max b2]`

- Sets with associative operations are called semigroups.

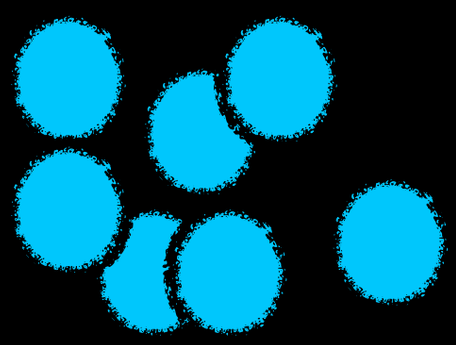
- With a special 0 such that 0+a=a +0=a for all a, they are called monoids.

- Many computations are associative, or can be expressed that way.

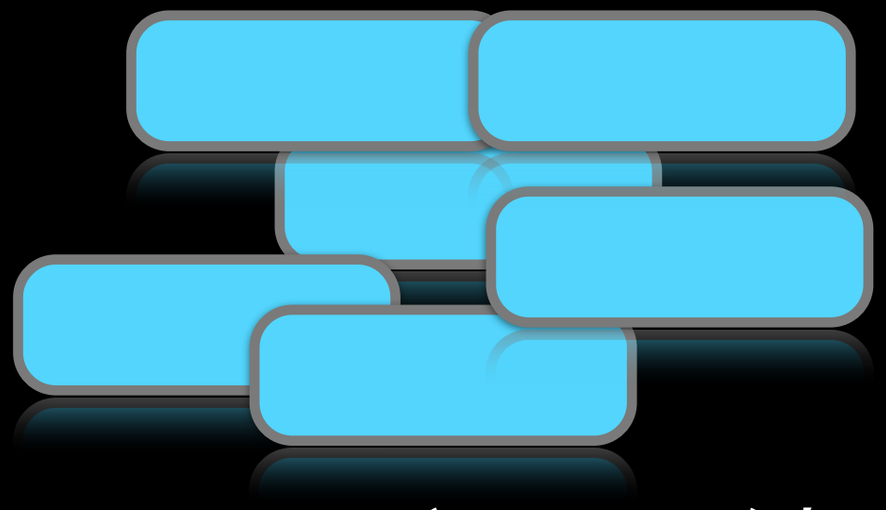
- Lack of associativity increases latency exponentially.

Part 2: Hash, don't sample

Problem: show cool tweets, don't repeat.

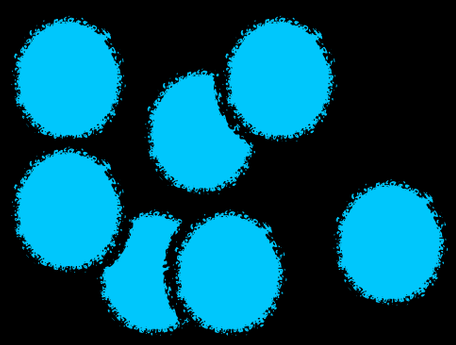


Users (>10^8)

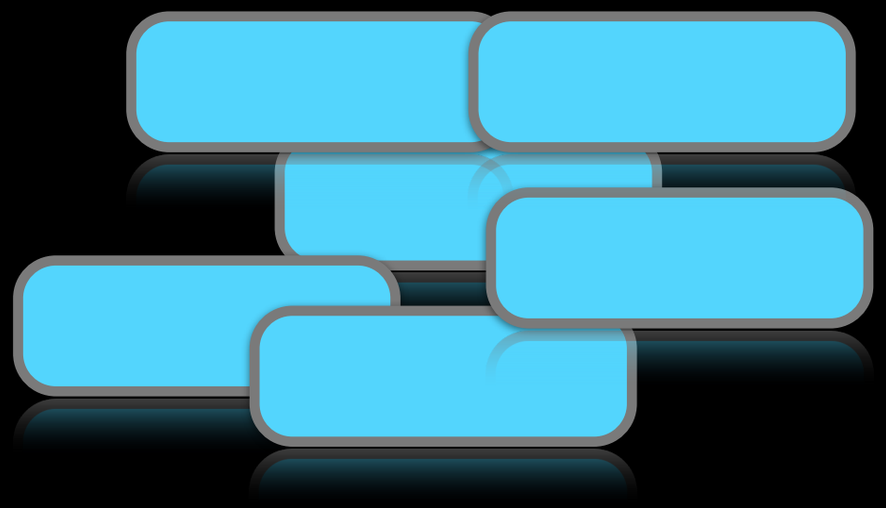


Tweets (>10^8/day)

Problem: show cool tweets, don't repeat.



Users (>10^8)



Tweets (>10^8/day)

Storing the graph (u -> t) as a Set[(U,T)] or Map[U, Set[T]] takes a lot of space, costly to transfer, etc.

Solution: Bloom Filter

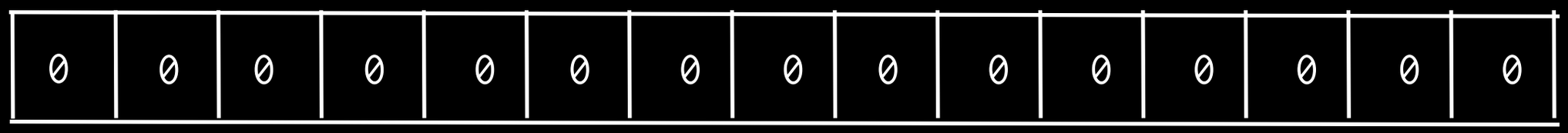
- Like an approximate Set
- Bloom.contains(x) => Maybe|No
- Prob false positive > 0.
- Prob false negative = 0.

Bloom Filter

We want to store i in our set:

i

m-bit array



Bloom Filter

i

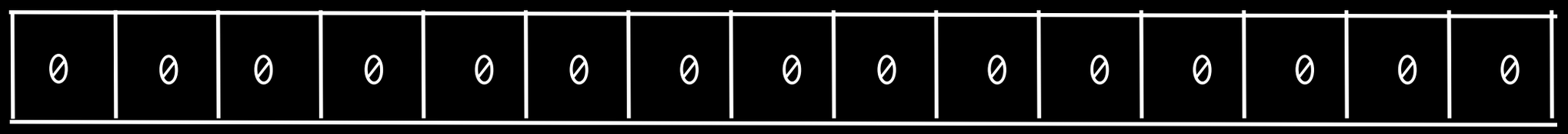
k hashes
=>[1,m]

hash3(i)=14

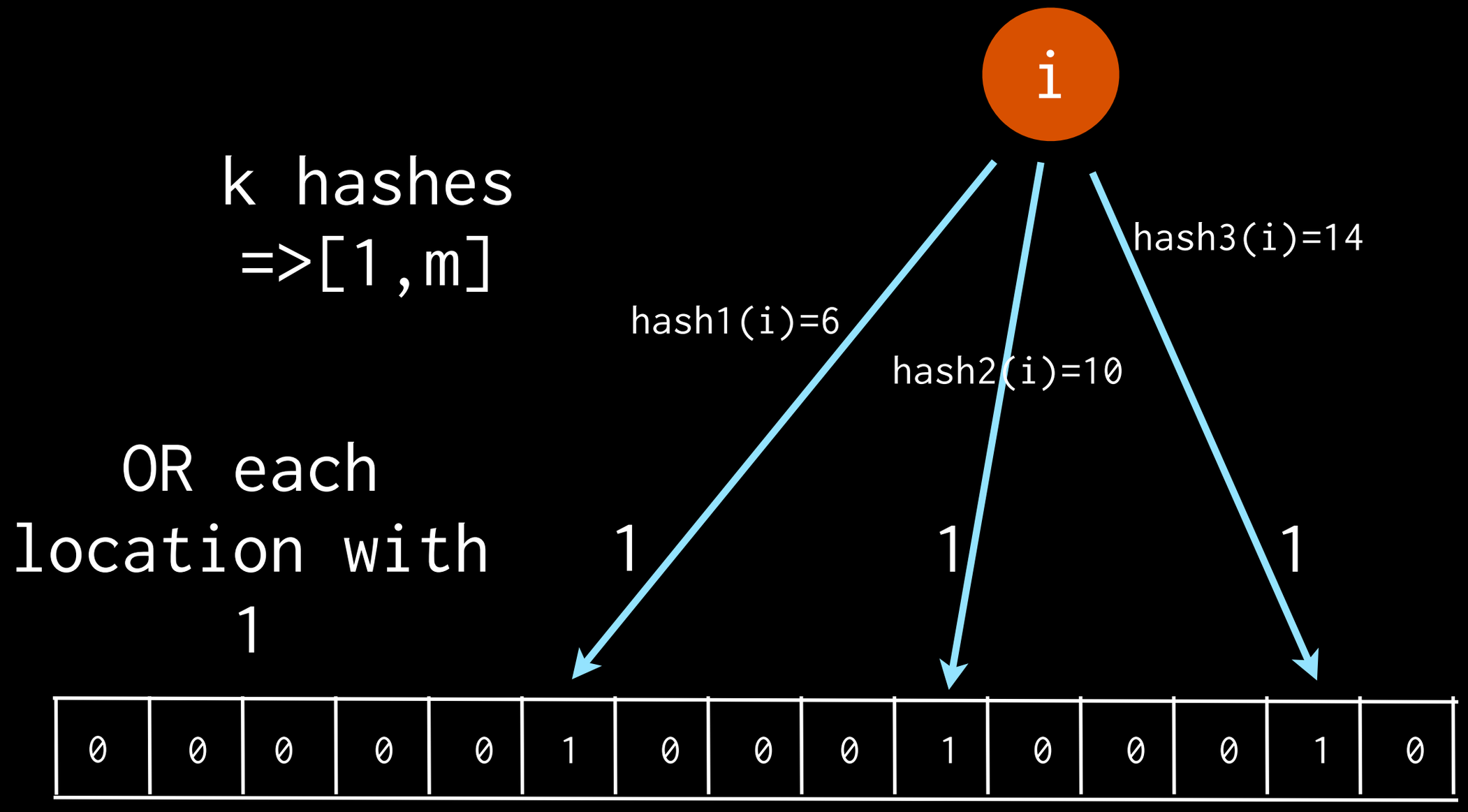
hash1(i)=6

hash2(i)=10

m-bit array

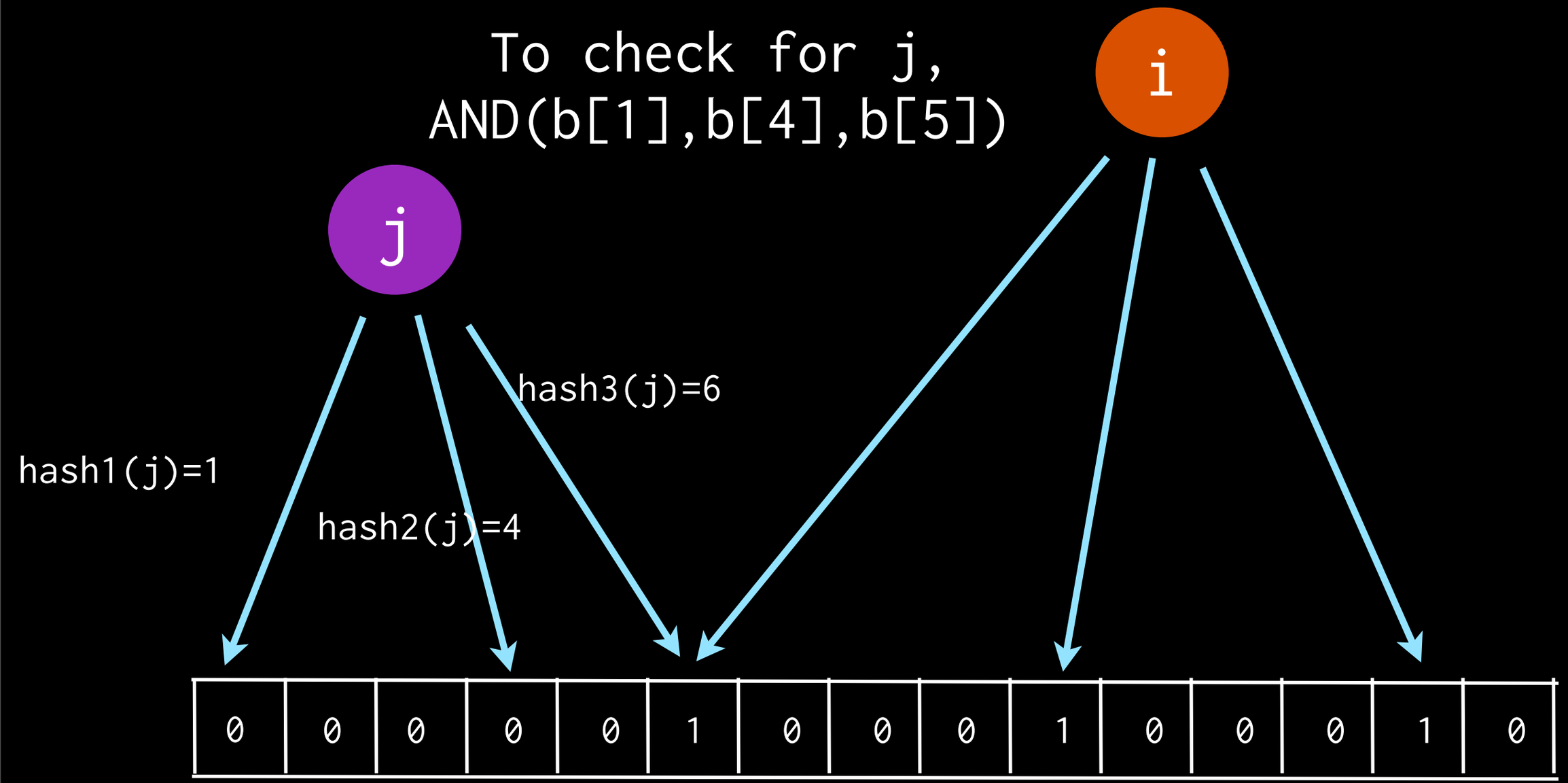


Bloom Filter



Bloom Filter

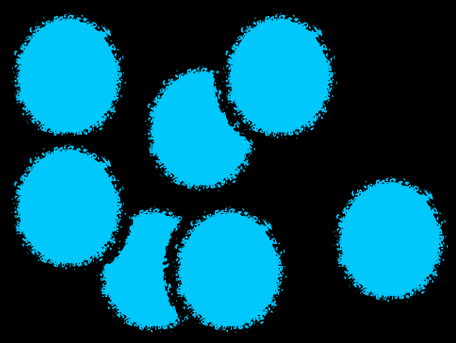
To check for j,
AND(b[1],b[4],b[5])



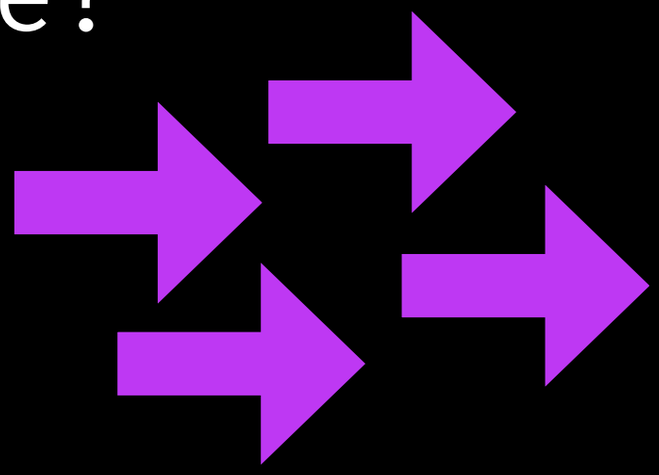
What's going on

- hash to a set of indices, OR those with 1, read by taking AND.
- writing uses boolean OR, that's a monoid, so we can do this in parallel => lowers latency. Reading also a monoid (AND)!
- We can tune false prob by tuning m(bits) and k(hashes),
- p~exp(-m/(2n)) for n items, k=0.7m/n

Problem: how many unique users take all pairs of actions on the site?



Users (>10^8)



Actions (look at Tweet x, follow user y, etc...)

To count Set size, we may need to store the whole set (maybe all users?) for all these pairs of actions (HUGE!)

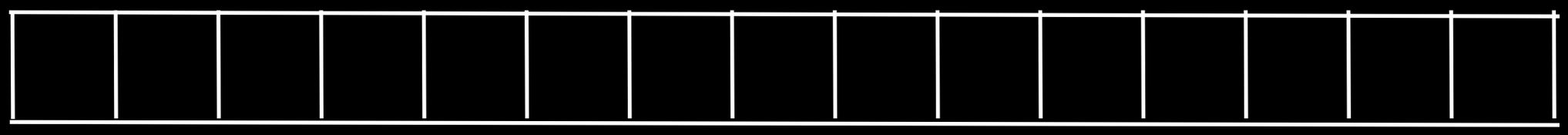
Solution: HyperLogLog

- Like an approximate Set
- HLL.size => Approx[Number]
- We know a distribution on the error.

Hyperloglog

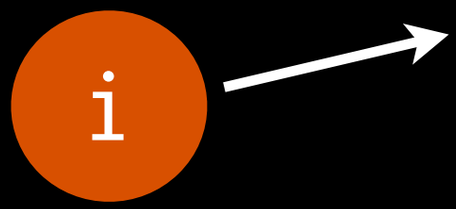
User i takes an action, we want to add to our approximate set:

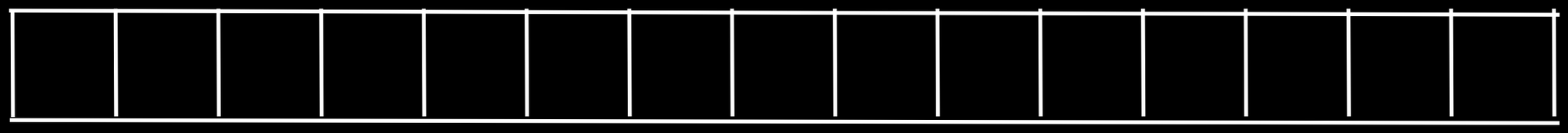
i



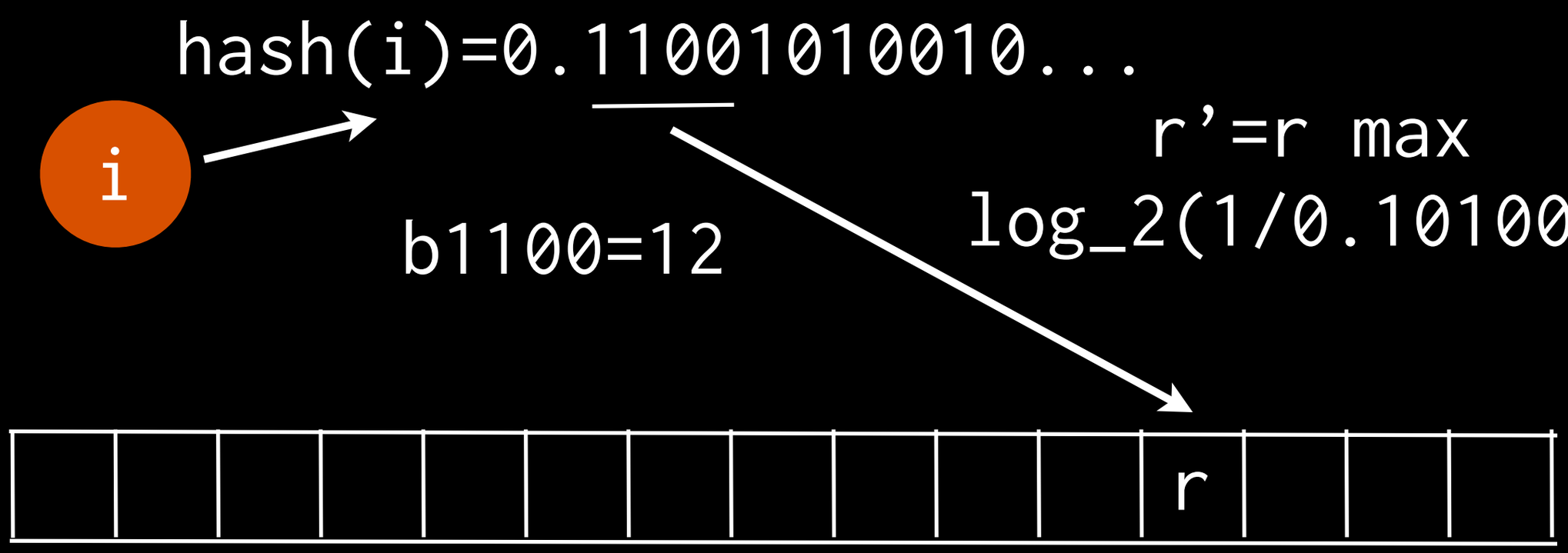
Hyperloglog

hash(i)=0.11001010010...





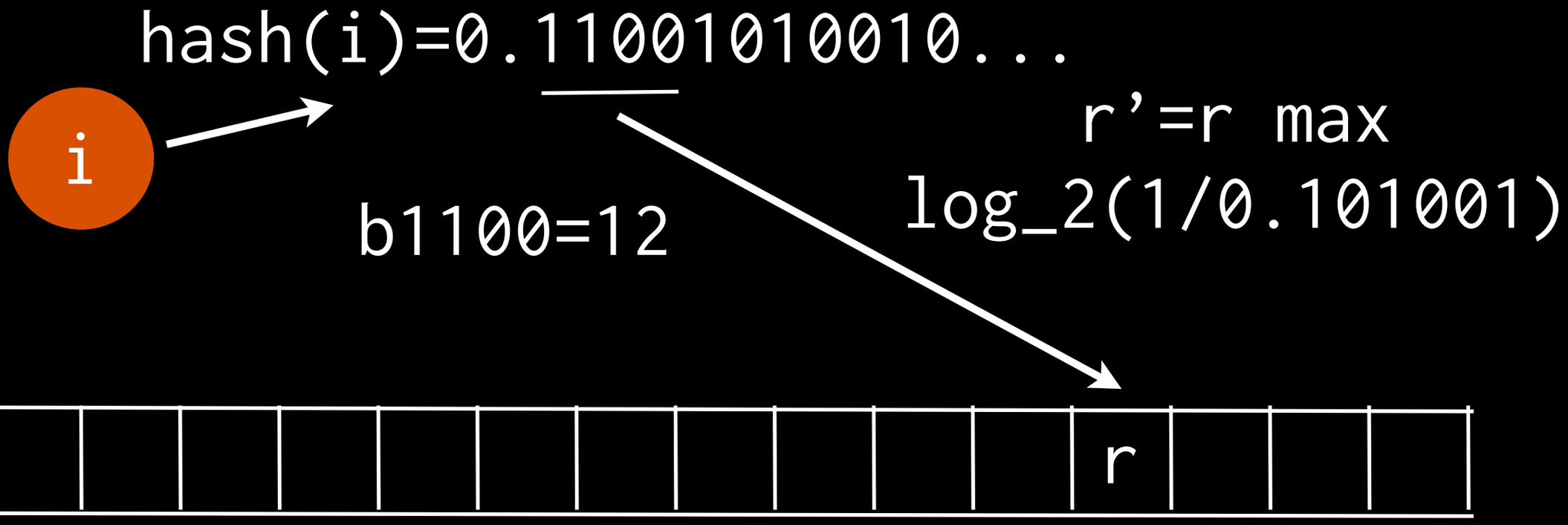
Hyperloglog



a_m m^2/Estimate = sum(1/2^r)

(where a_m is some normalizing constant).

Hyperloglog



Intuition: Each bucket holds max of ~1/m values,
so each bucket estimates size: S/m ~ 2^r
Harmonic mean estimates total size ~
1/(1/m sum(1/(m2^r)))

What's going on

in HyperLogLog

- hash to 1 index and value r, MAX that with existing, read by taking HARMONIC_SUM of all buckets.

- writing uses MAX, that's a monoid, so we can do this in parallel => lowers latency. reading also uses monoid! (HARMONIC_SUM)
- We can tune size error by tuning bucket count (m) and bits used to store r.
- std. error ~ 1.04/sqrt(m)

It's (monoidal)
deja vu all over
again

Remember:

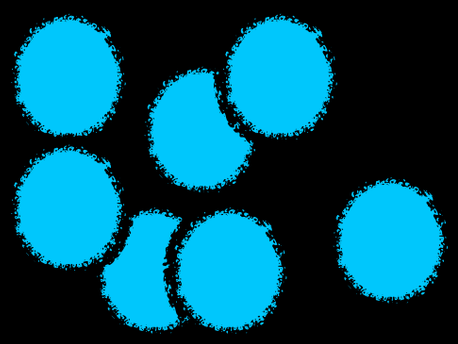
What's going on

in Bloomfilter

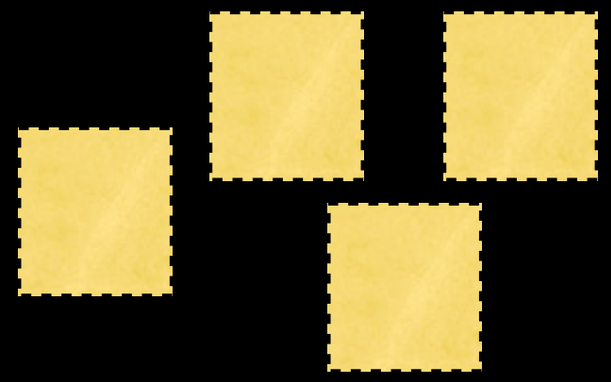
- hash to a set of indices, OR those with 1, read by taking AND.
- writing uses boolean OR, that's a monoid, so we can do this in parallel => lowers latency. Reading also a monoid (AND)!
- We can tune false prob by tuning m(bits) and k(hashes),
- p~exp(-m/(2n)) for n items, k=0.7m/n

What else looks like this?

Problem: How many tweets did each user make on each hour?



Users (>10^8)

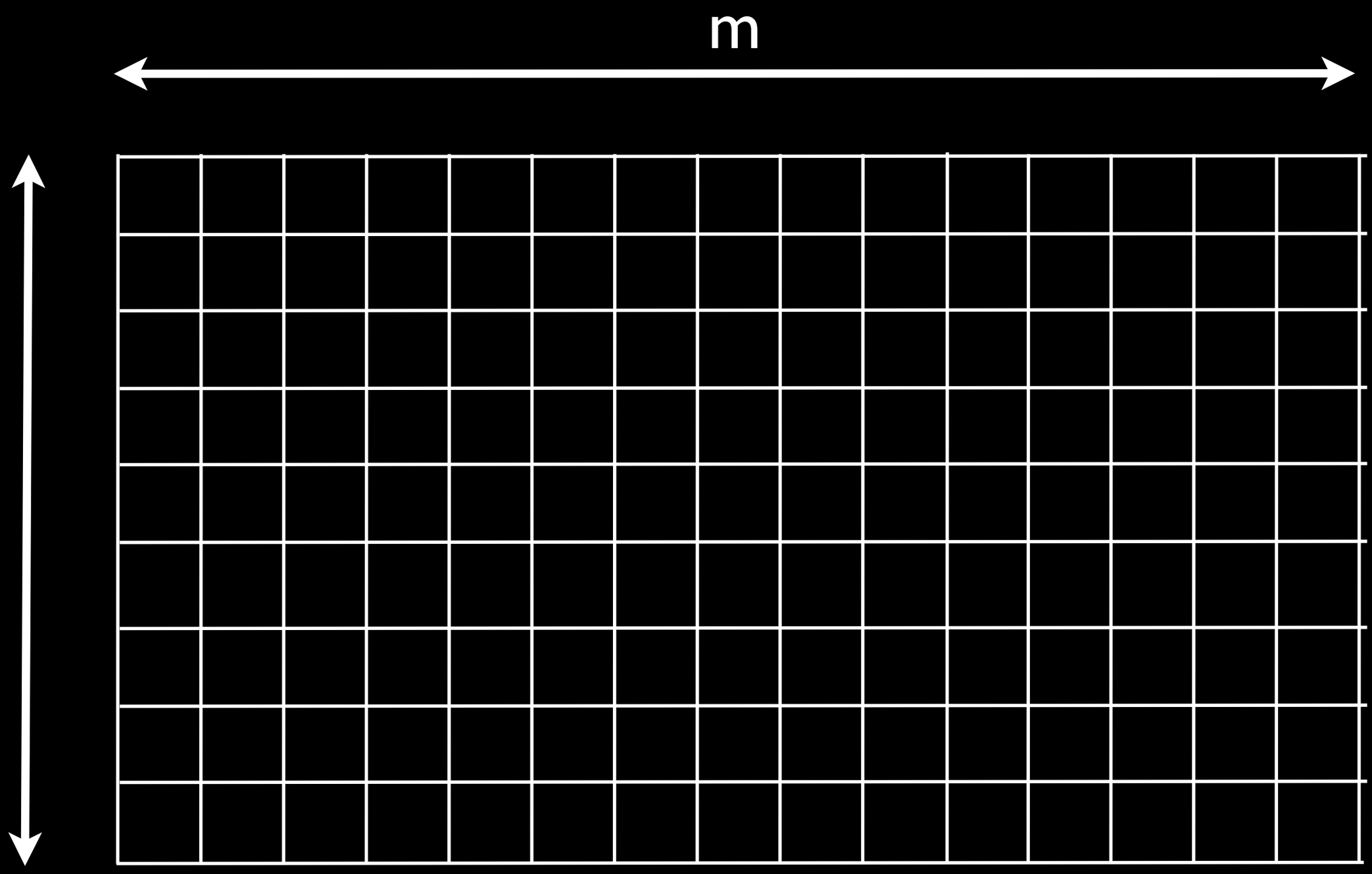


196 hours/week x 52 weeks/ year x 7 years of tweets

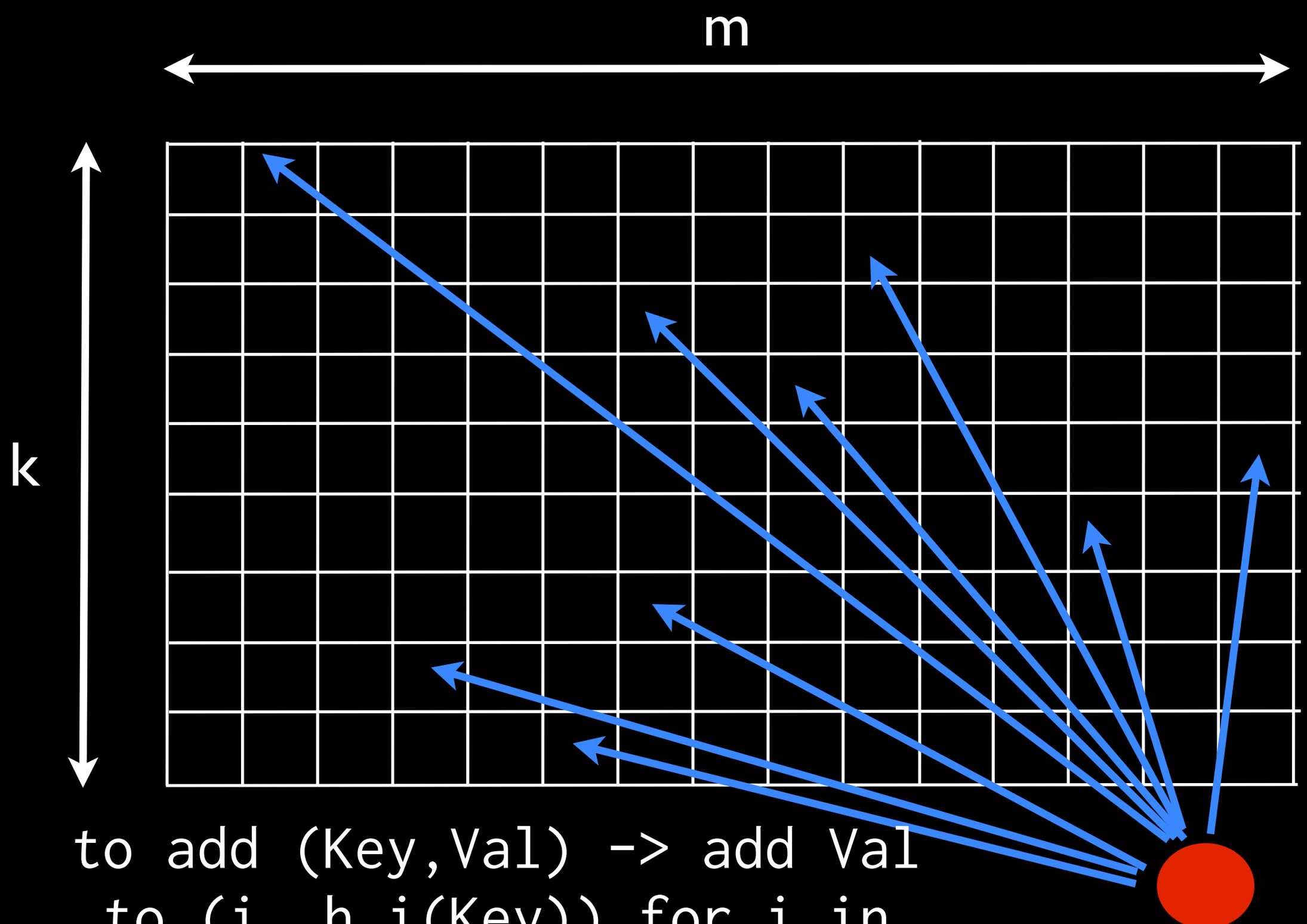
If we make a key for each (user, hour) pair we have 10s of trillions potential keys

Solution: Count-Min Sketch

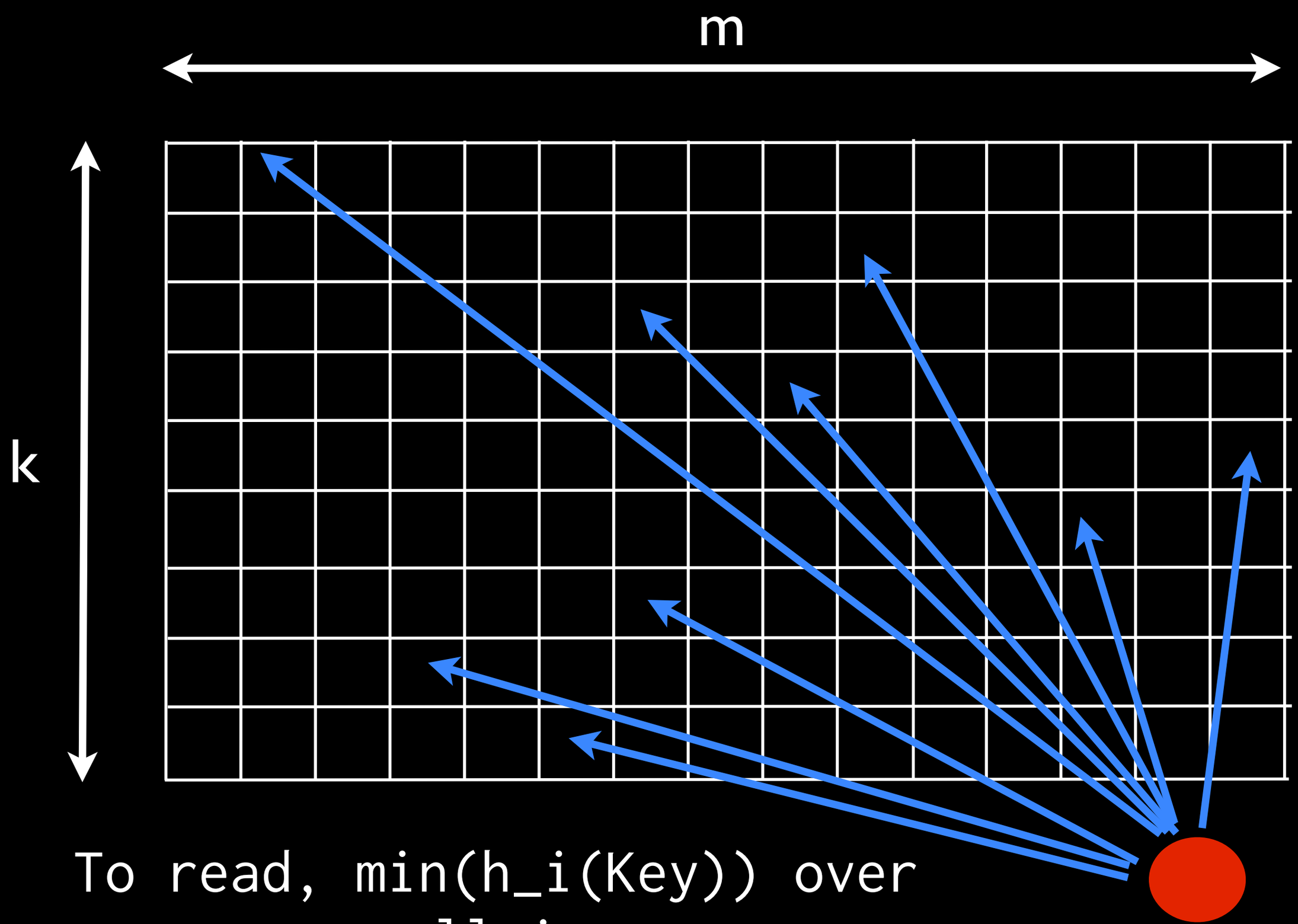
- Like an approximate Counter or Map[K, Number]
- CMS.get(key) => Approx[Number]
- It always returns an upper bound, but may overestimate (we know the control the error).



We have k hash functions
onto a space of size m




```
to add (Key,Val) -> add Val
 to (i, h_i(Key)) for i in
              (1,k)
```



To read, min(h_i(Key)) over all i.

What's going on

in Count-Min-Sketch

- hash to a set of indices, ADD those with 1, read by taking MIN.
- writing uses numeric ADD, that's a monoid, so we can do this in parallel => lowers latency. Reading also a monoid (MIN)!
- We can tune error: Prob > 1 - delta, error is at most eps * (Total Count).
- m = 1/eps, k = log(1/delta)

	Hashes	Write Monoid	Read Monoid
Bloom Filter	k-hashes into 1 m-dim binary space, read same hashes.	Boolean OR	Boolean AND
HyperLogLog	1-hash into m dimensional real space, read whole space.	Numeric MAX	Harmonic Sum
Count-min-sketch	d-hashes onto d non-overlapping m dimensional spaces, read same hashes.	Numeric Sum	Numeric MIN

- All use hashing to prepare some vector.
- The values are always Ordered (bools, reals, integers).
- These monoids are all commutative.
- The write monoid has: a + b >= a, b
- The read monoid has: a + b <= a, b

Summary: Why Hashing

- We can model hashed data structures as Sets, Maps, etc... familiar to programmers => accessibility.
- Sampling in complex computations is hard! How to sample correlated events (edges in graphs, communities, etc...) hashing can sidestep but still be on a budget.
- Hash-sketches are naturally are Monoids, and thus are highly efficient for map/reduce or streaming applications.

Call to Arms!

- Many sketch/hashes are less than 10 years old. Lots to do!
- There is clearly something general going on here, what is the larger theory than describes all of this?
- Sketches can be composed, which allows non-experts to leverage them.
- Sketches often have properties amenable to parallelization (Monoids)!

Algebird

- http://github.com/twitter/algebird
- baked in to summingbird, scalding and examples for spark.
- Implementations of all the monoids here, and many more.

- Tons O' Monoids:
- CMS, HyperLogLog, ExponentialMA, BloomFilter, Moments, MinHash, TopK

https://github.com/twitter/algebird/tree/develop/algebird-core/src/main/scala/com/twitter/algebird

d Libra | RB | My Delicious | SavePublishing | Bookmark on Delicio | Add to Wish List | Twit

algebird / algebird-core / src / main / scala / com / twitter / **algebird** /

Merge pull request #136 from ccsevers/add_foldM ···

johnynek authored 3 days ago

..		
mutable	a month ago	Adds priority queue aggregator [johnynek]
AdjoinedUnitRing.scala	18 days ago	Cleans up intTimes [johnynek]
AffineFunction.scala	a month ago	test [sritchie]
Aggregator.scala	a month ago	test [sritchie]
Approximate.scala	a month ago	test [sritchie]
AveragedValue.scala	a month ago	test [sritchie]
BloomFilter.scala	a month ago	test [sritchie]
CountMinSketch.scala	3 days ago	Hotfix for CMS [johnynek]
DecayedValue.scala	a month ago	test [sritchie]
DecayedVector.scala	a month ago	test [sritchie]
Eventually.scala	a month ago	add eventually [sritchie]
Field.scala	a month ago	test [sritchie]
GeneratedAbstractAlgebra.scala	a month ago	test [sritchie]
Group.scala	18 days ago	Cleans up intTimes [johnynek]
HyperLogLog.scala	a month ago	test [sritchie]
IndexedSeq.scala	a month ago	test [sritchie]
JavaMonoids.scala	a month ago	test [sritchie]
MapAlgebra.scala	17 days ago	Adds a comment (to restart travis) [johnynek]
Metric.scala	a month ago	test [sritchie]
MinHasher.scala	a month ago	test [sritchie]

Follow

- @posco <-- me
- @scalding <-- easy Hadoop monoids!
- @summingbird <-- Monoids in realtime!

Thank you for coming